

System-in-Package Technology: Making it Easier to Build Your Own Linux Computer

Jason Kridner
Erik Welsh
03/12/2018



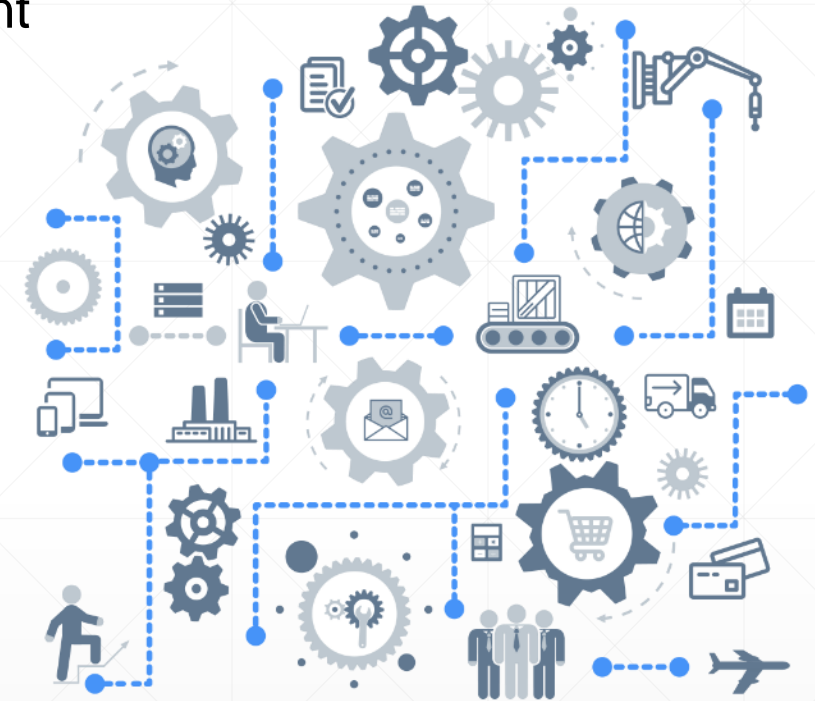
The SBC Prototyping Revolution

- Proliferation of prototyping boards
 - Huge array of Processors
 - Every kind of connector
 - Add-on boards for additional functionality
- Developing communities
 - Support for new users
 - Collaboration for experienced developers
- Exposure to Linux
 - Development of drivers
 - Open source projects



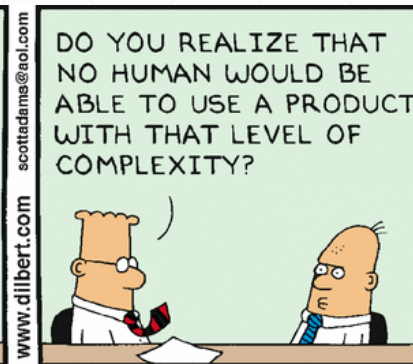
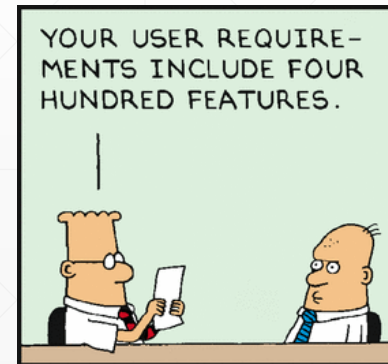
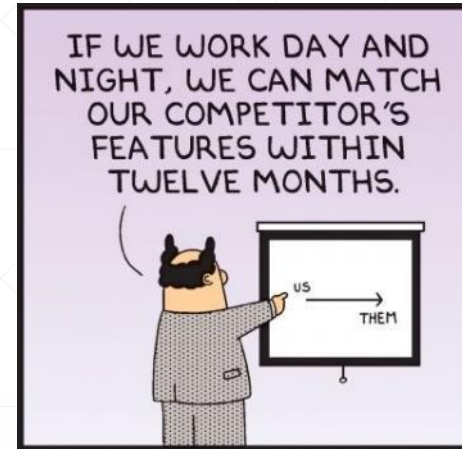
Software Drives Hardware Decisions

- Software developers must be involved in Hardware development
 - If there is no SW for a piece of HW, then don't use it
 - Many platforms provide great starting point for SW development
 - Focus on value added feature differentiation
 - Re-writing drivers does not add value
 - Don't allow changes in HW for the sake of changing HW
 - SW impact needs to be understood
- With great power, comes great responsibility ...
 - Choosing a platform with Open Hardware
 - Using components that can be obtained from Distribution / in small quantities
 - Hardware should also focus on value added features
 - Routing DDR does not add value



“Mind the Gap” Moving from Prototype to Product

- Developing custom PCB
 - Smaller is better ... but smaller is harder
 - Open hardware help development; known good solutions
- Migrating Software
 - Porting from development board to final components
 - Bring up & Provisioning
- Doing more with less
 - Smaller teams
 - Need tools that reduce time and effort



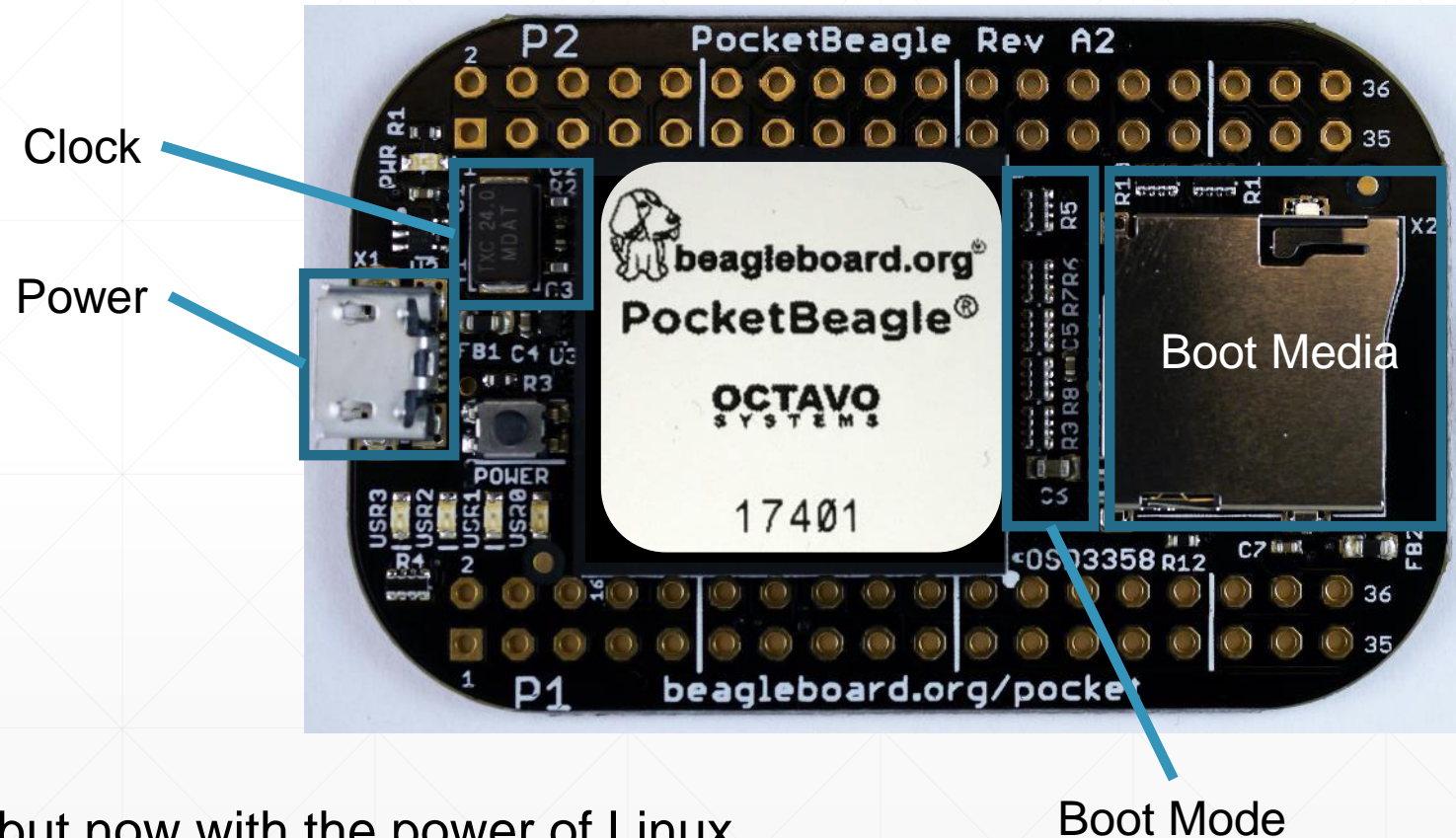
System-in-Package provides simple Linux HW Solutions

- Minimum Hardware Required to Run Linux

- Connect power inputs
- Connect clock inputs
- Select boot mode
- Provide Linux boot image

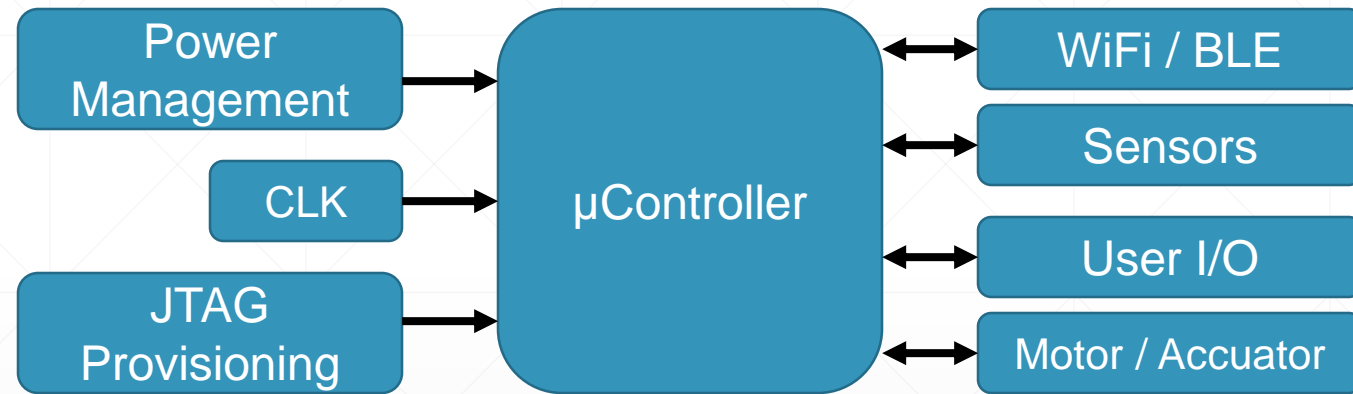
- Proven Linux solution

- Like working with a microcontroller but now with the power of Linux



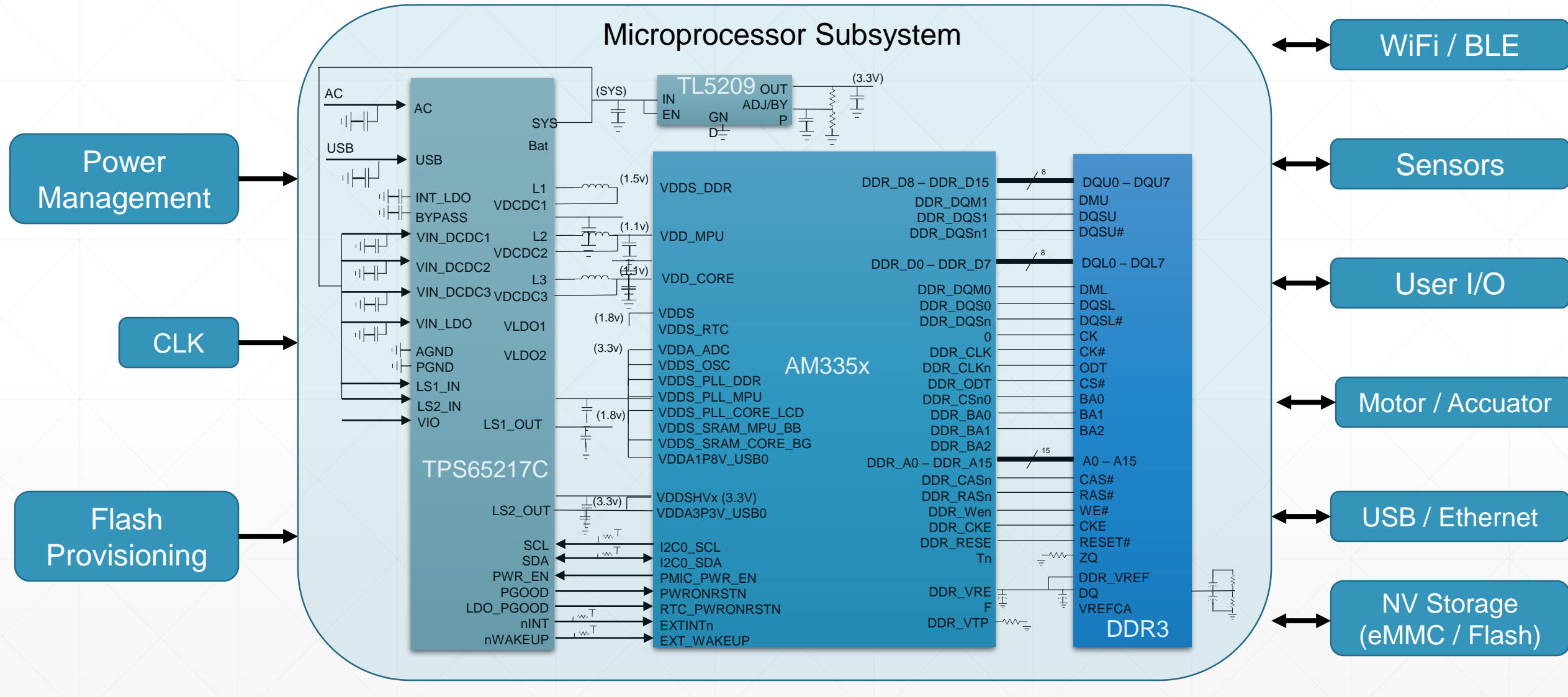
Moving up from a microcontroller can be scary ...

Microcontroller System Block Diagram



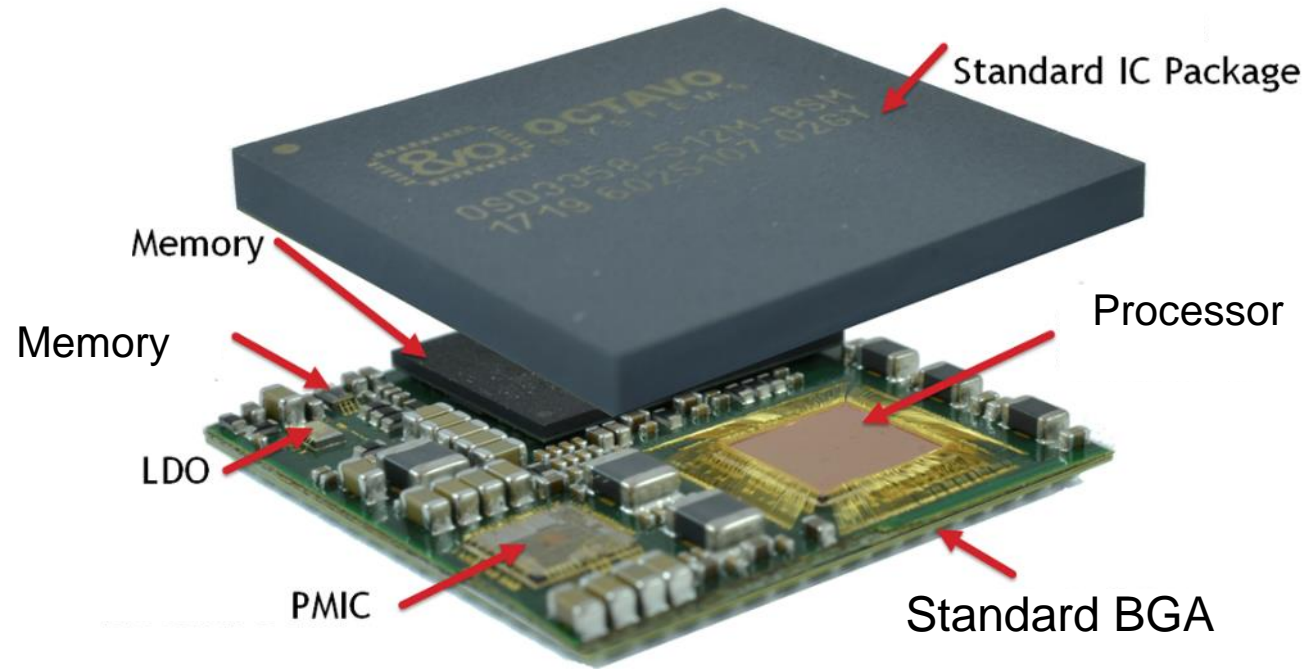
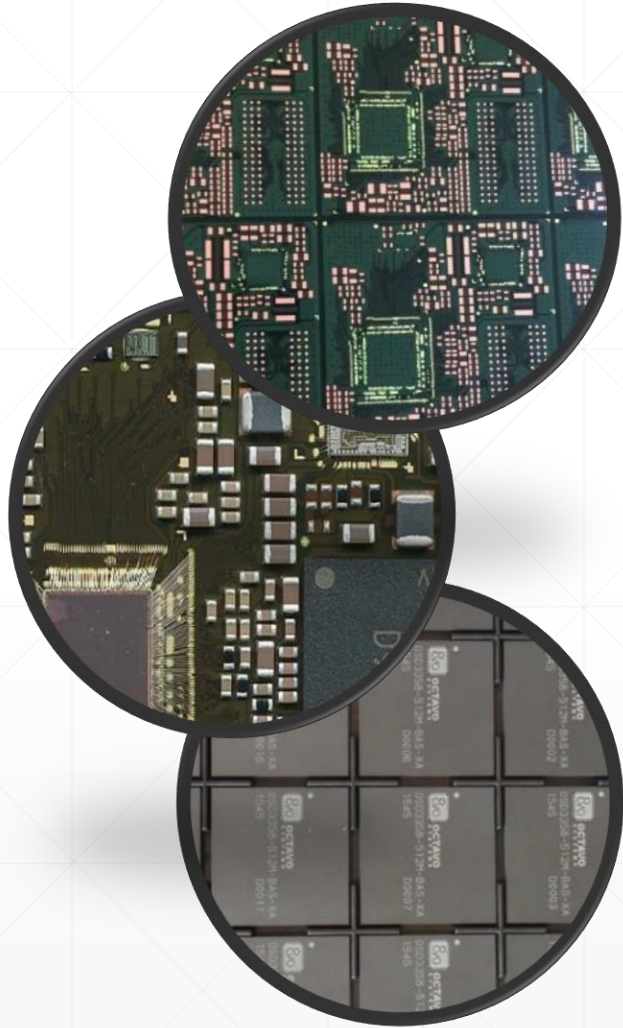
Becomes

Typical Microprocessor System Block Diagram

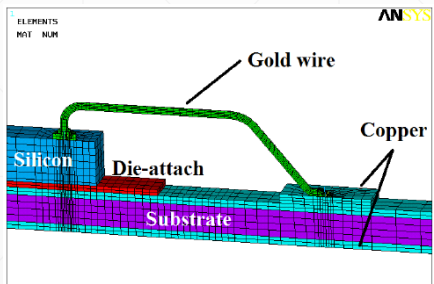
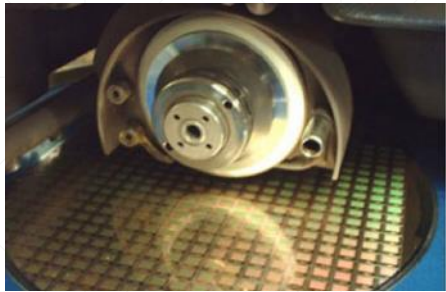
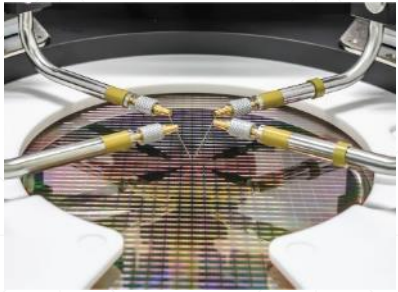


How can we simplify this complexity?

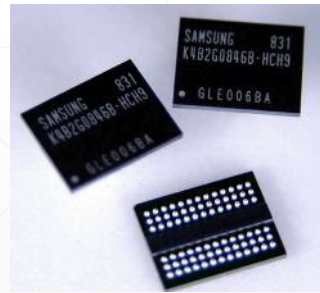
System-in-Package



What is System-in-Package



Attached Die



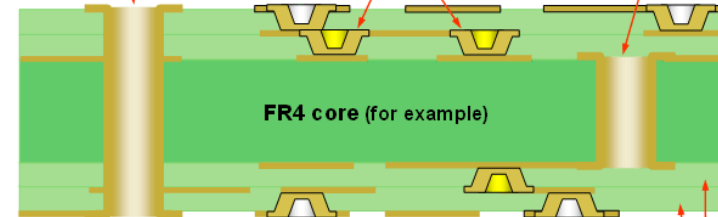
Discrete Components



A traditional drilled-right-through via hole

'Buried' microvias

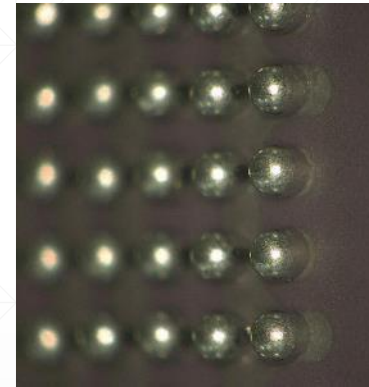
A drilled via on the inner FR4 core only



'Blind' microvias

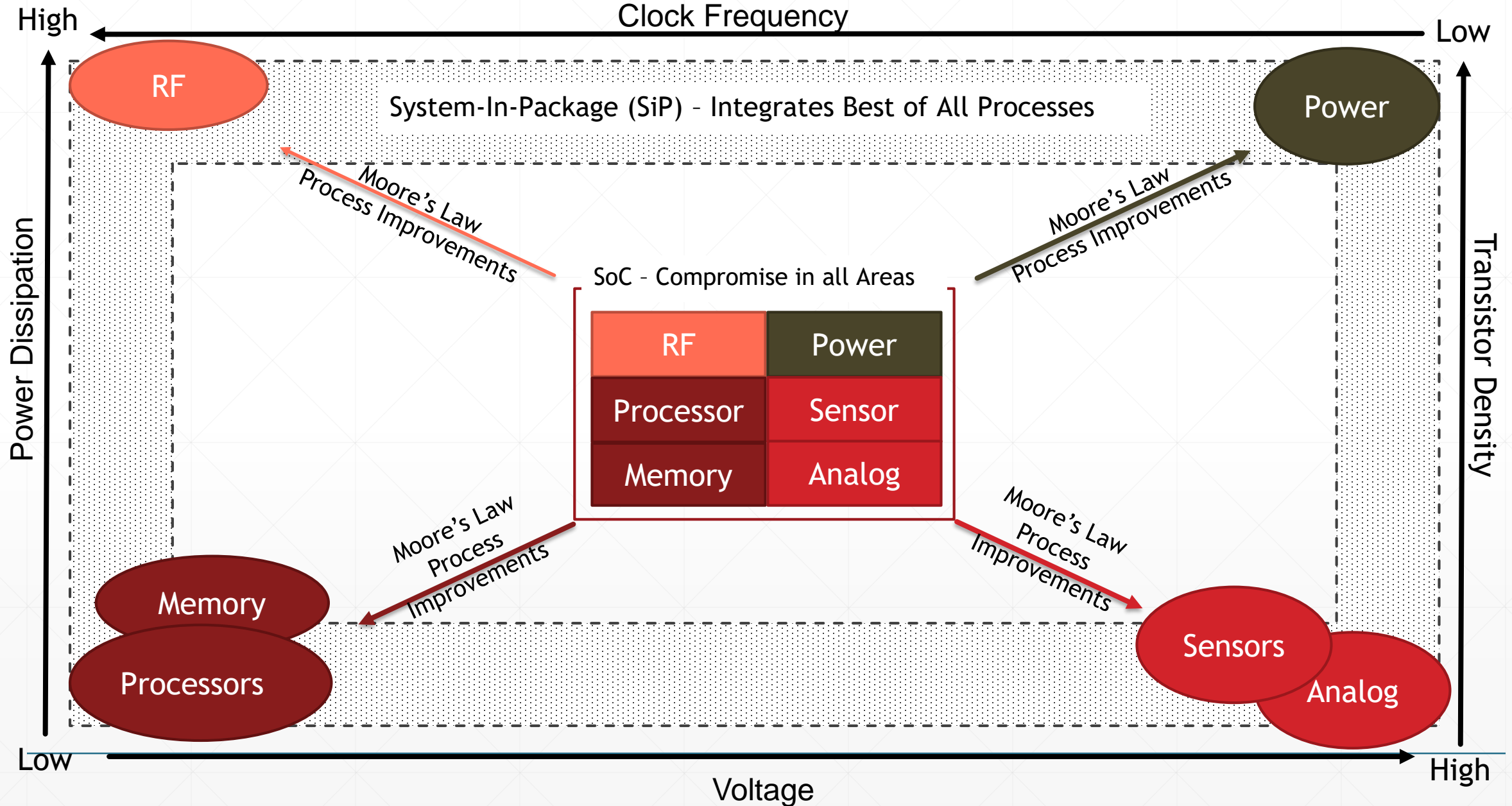
'Build-up' layers of pure polymer

Substrate

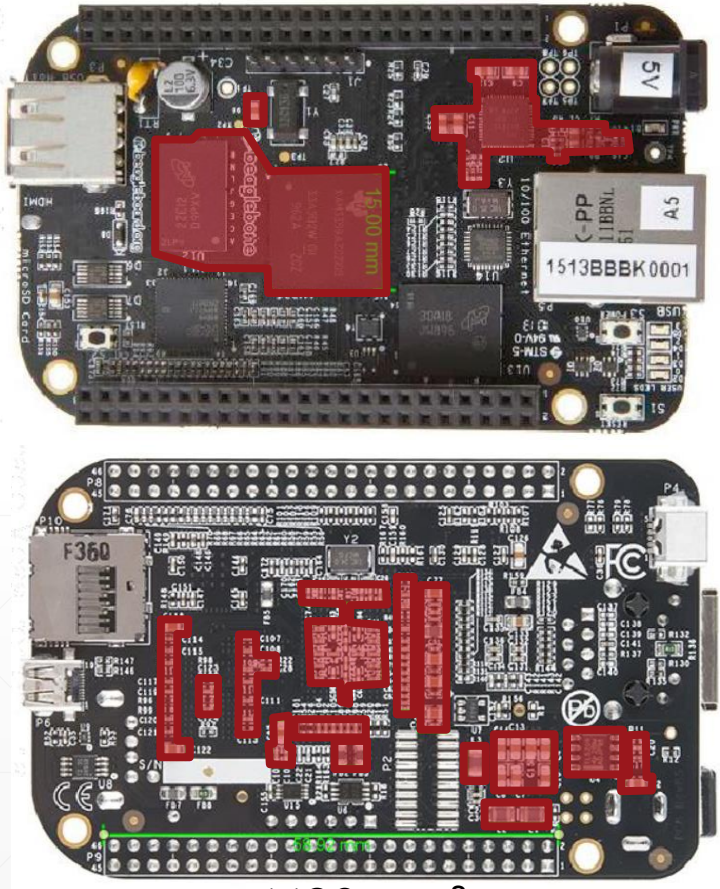


Pins

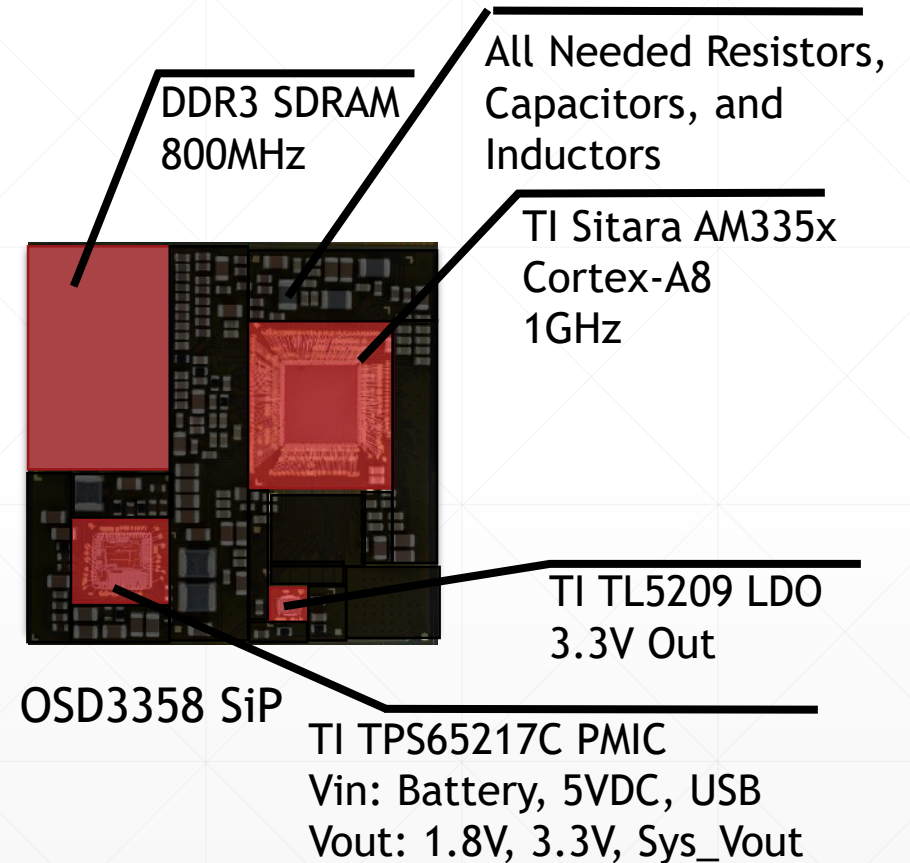
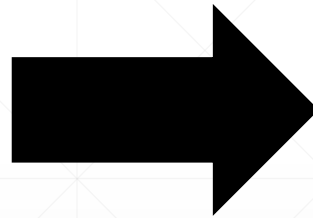
Why Can't We Just Use an SoC?



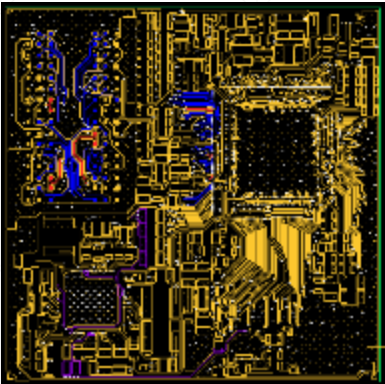
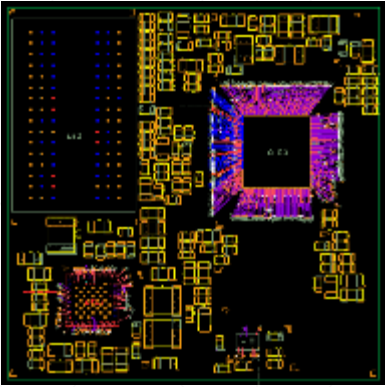
OSD3358 SiP Integration



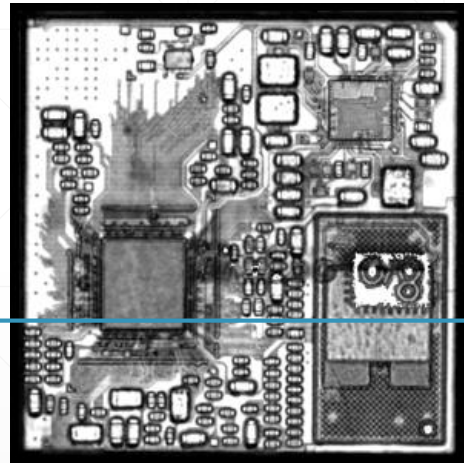
1130 mm²
BeagleBone Black Board



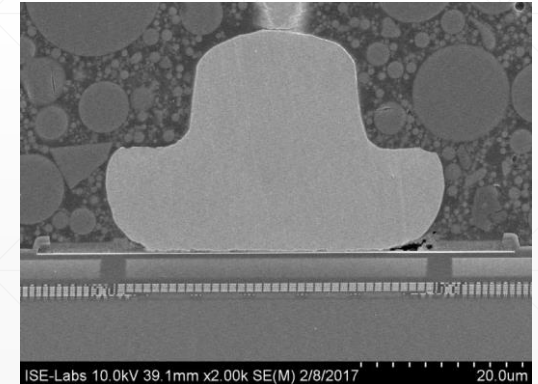
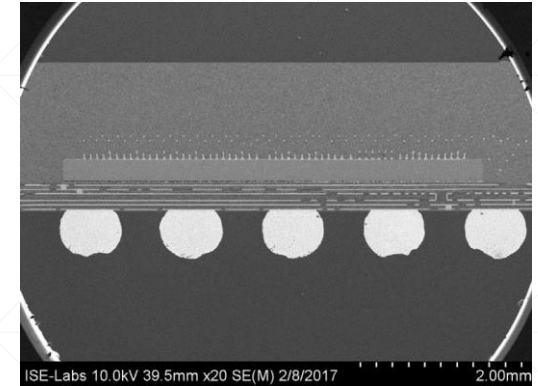
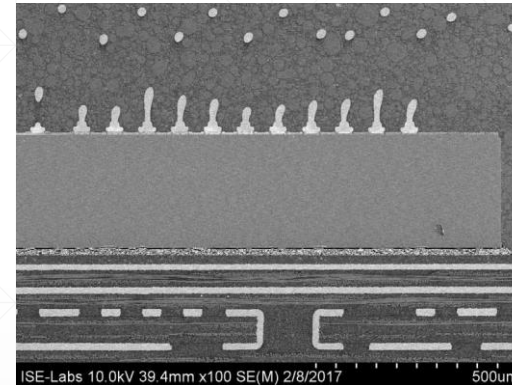
A Closer Look at a SiP



6 Layer Substrate



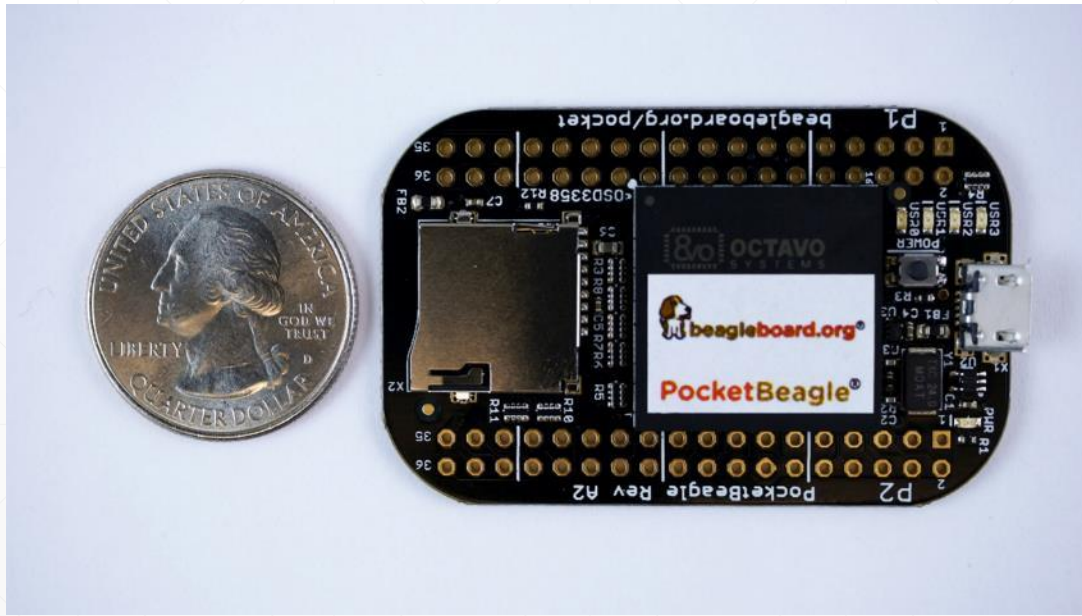
Manufactured SiP



Cross-Section SEM Picture of SiP

PocketBeagle

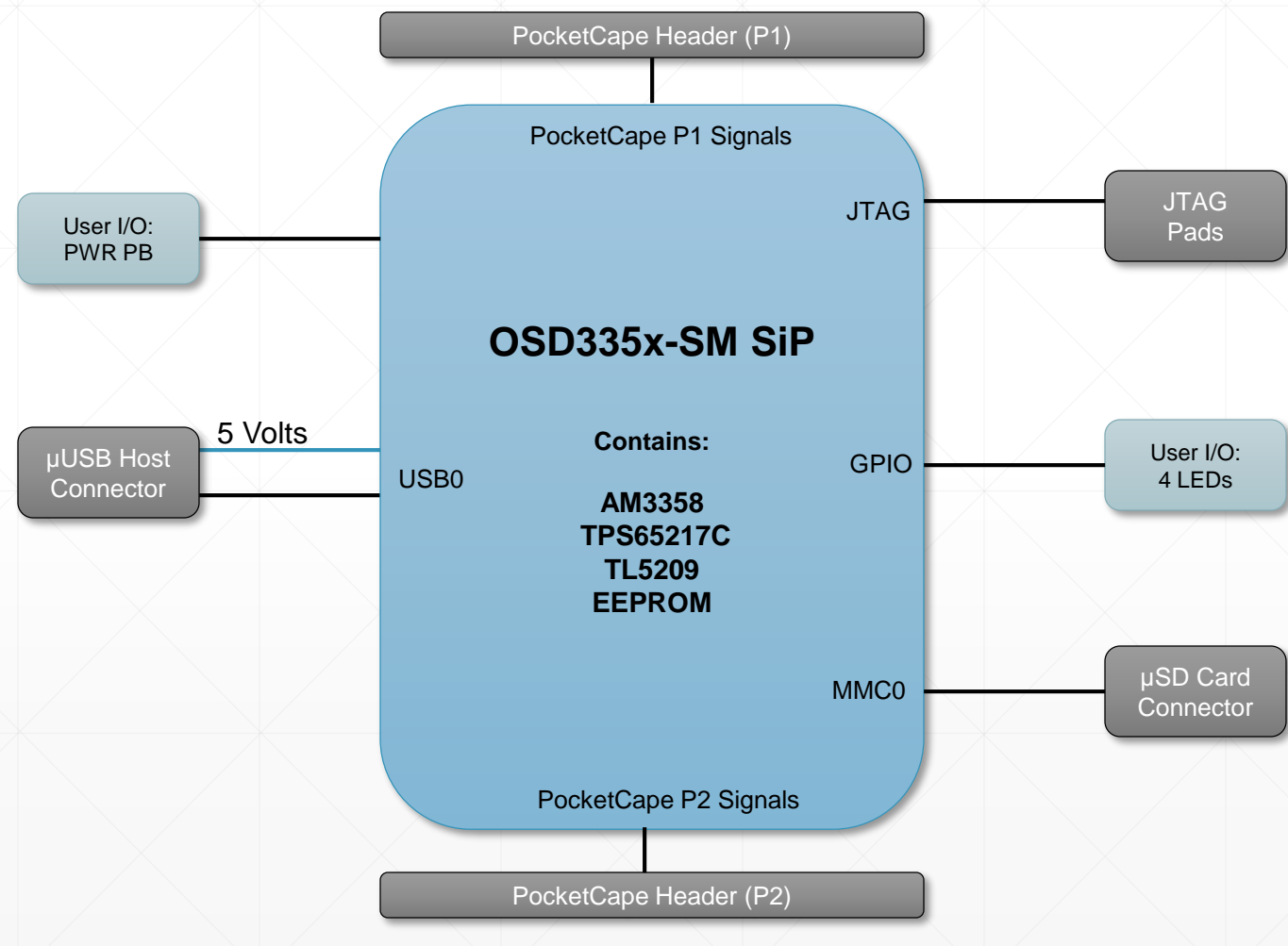
- <http://bbb.io/pocket>
 - Forums: <http://bbb.io/discuss>
 - News: <http://bbb.io/news>



- Based on Octavo Systems OSD3358-SM SiP
 - ARM Cortex-A8 @ 1-GHz
 - 512 MB DDRs RAM integrated
 - ARM Cortex-M3
 - 2x200-MHz RISC Programmable Real-time Units (PRU)
 - Integrated power management
- Connectivity
 - Bootable microSD card slot
 - High speed USB 2.0 OTG (host/client) control signals
 - Dual 36-pin expansion headers
 - 8 analog inputs (6 @ 1.8V and 2 @ 3.3V)
 - 44 digital GPIOs
 - 3 UARTS
 - 2 I2C
 - 2 SPI
 - 4 PWM
 - 2 QEP
 - 2 CAN
- \$25

56mm x 35mm x 5mm

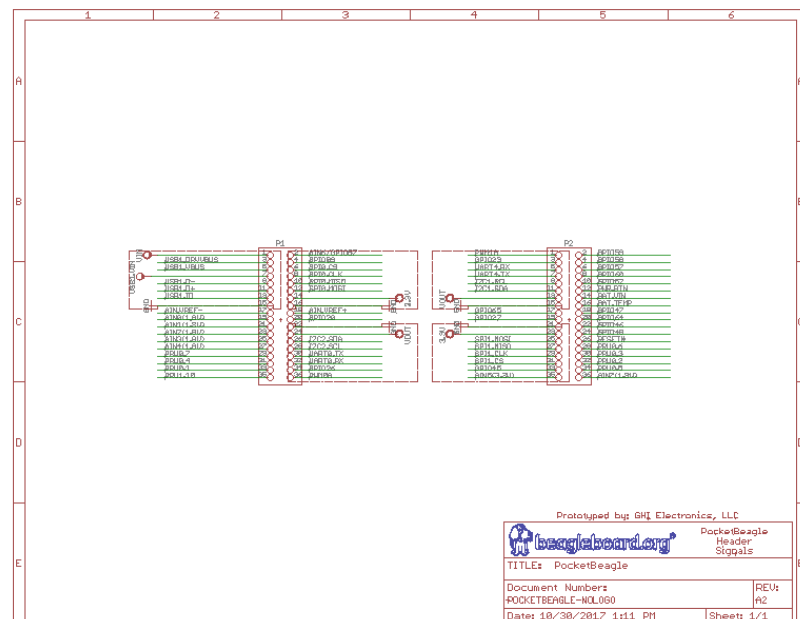
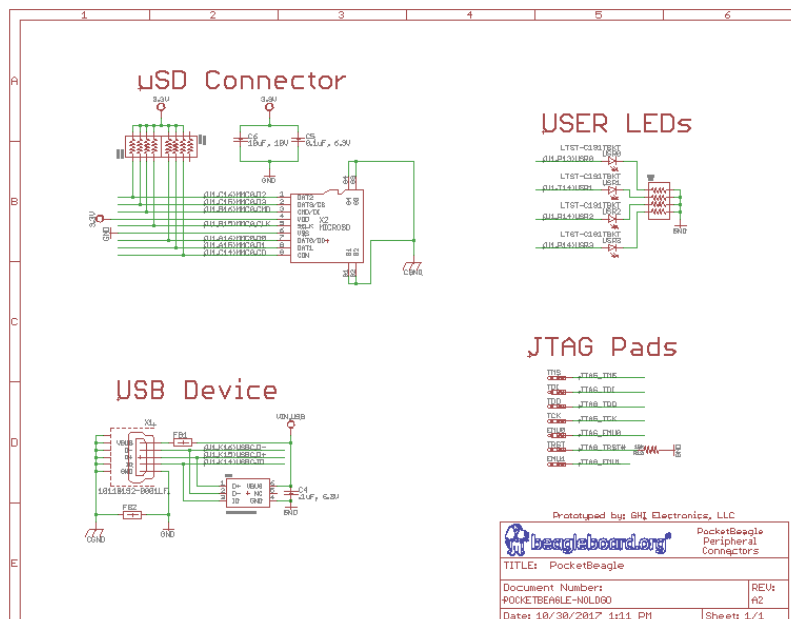
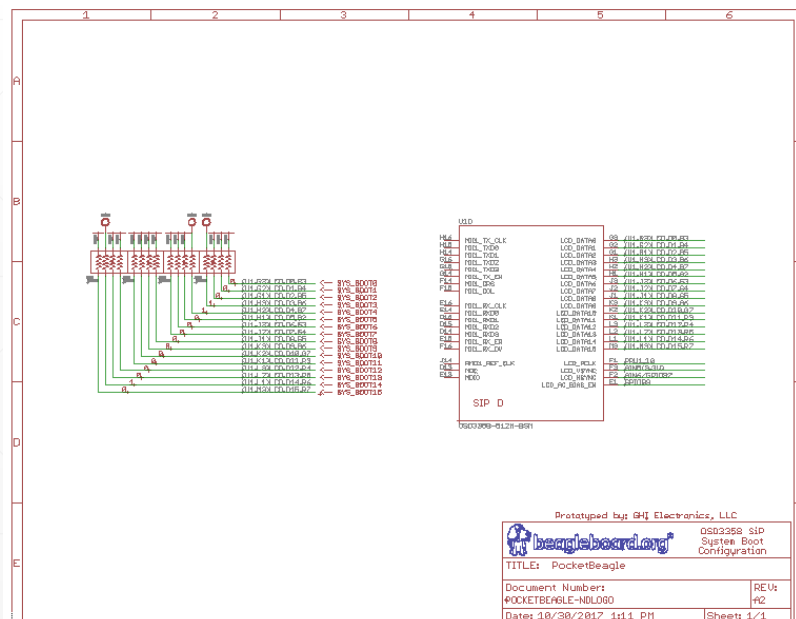
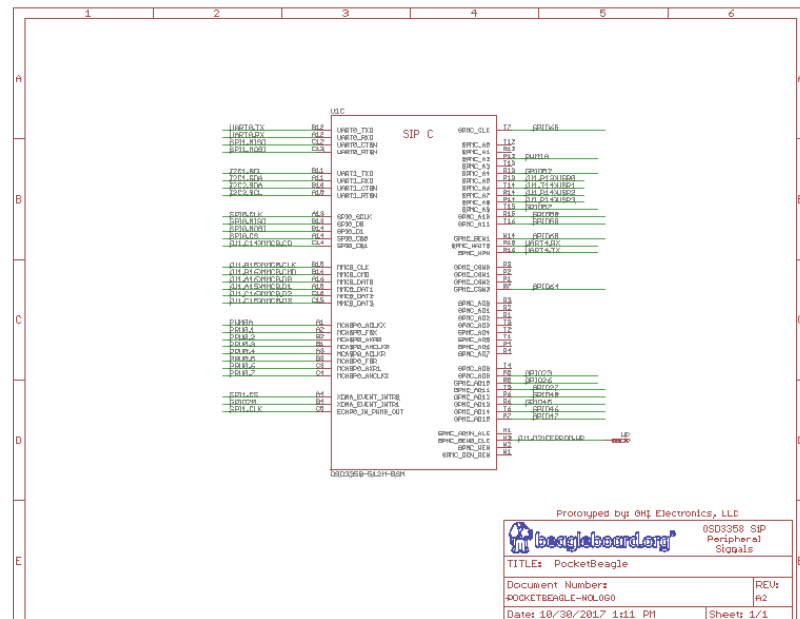
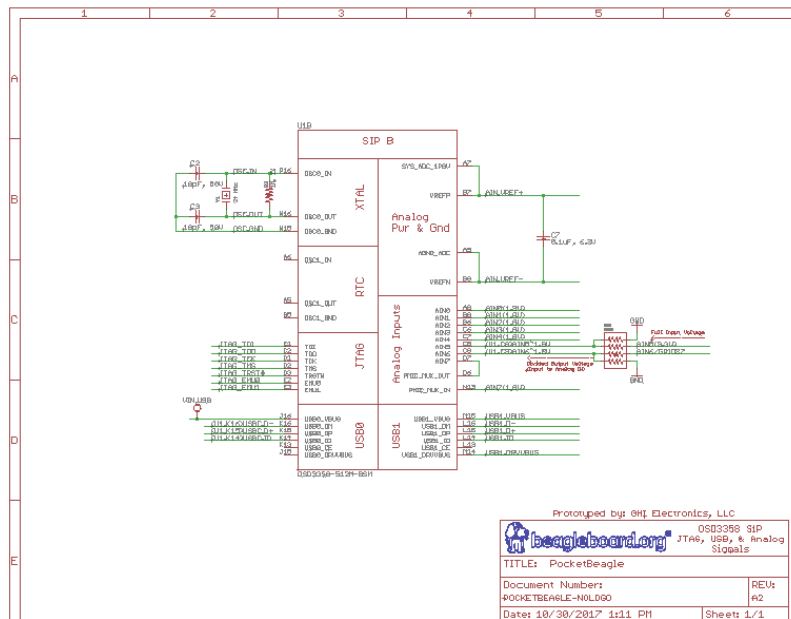
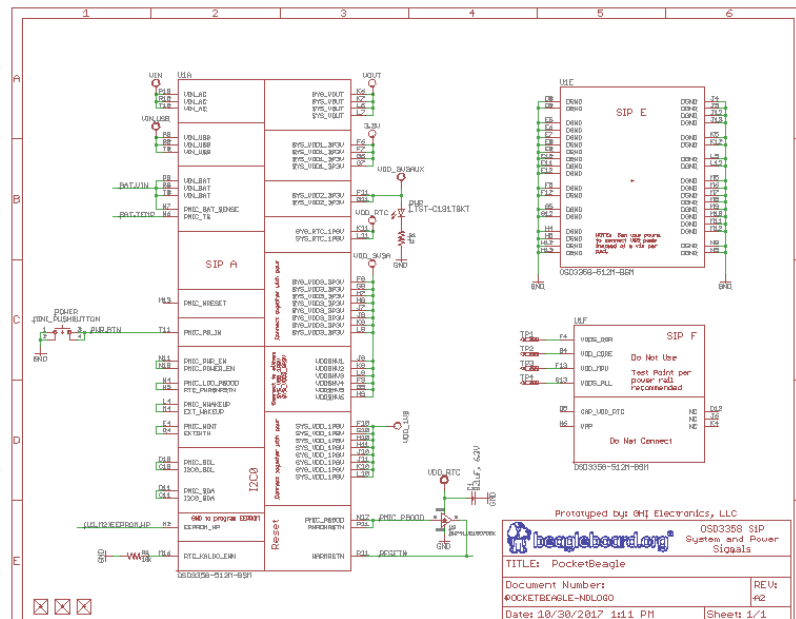
PocketBeagle Block Diagram



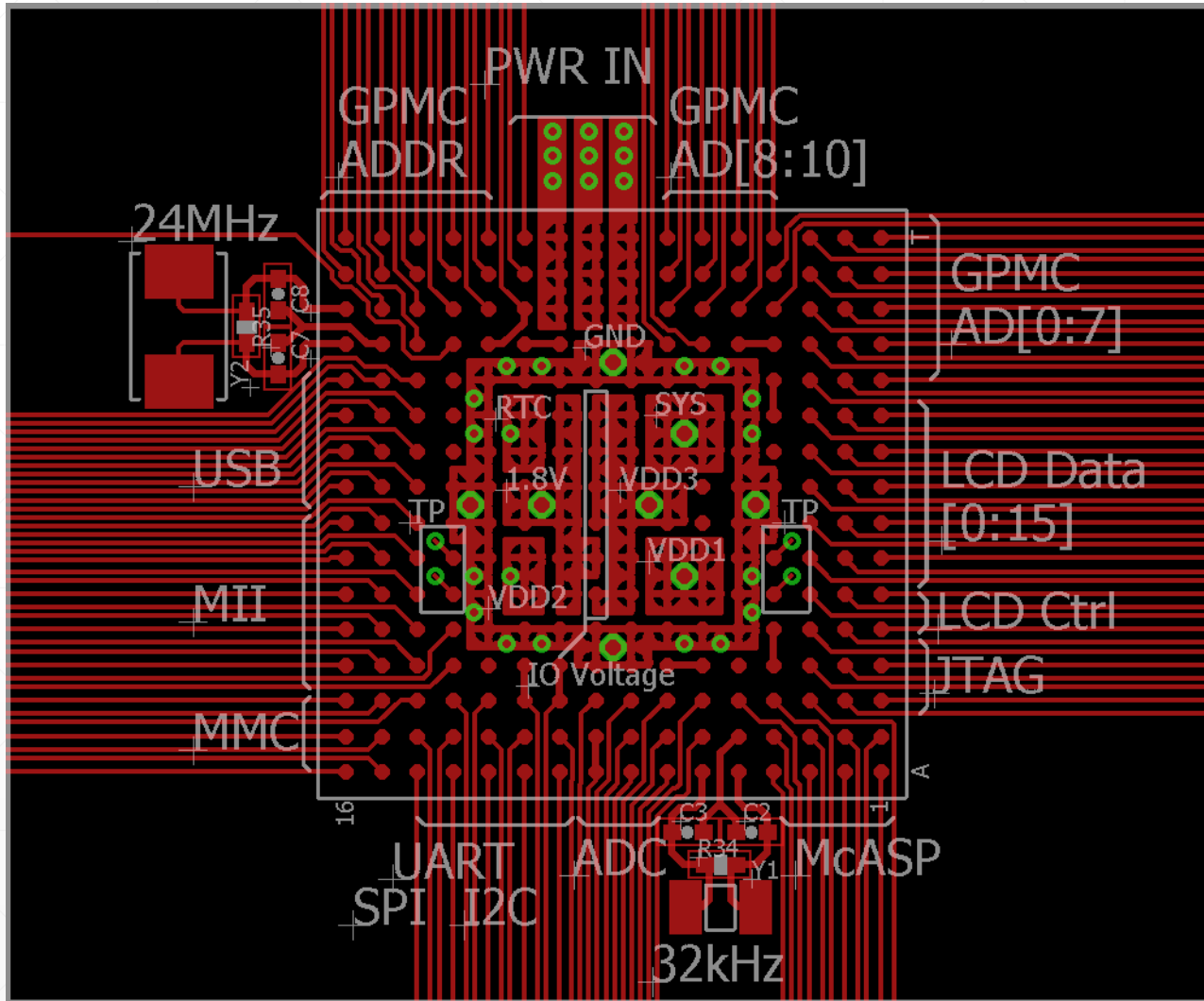
P1																
SYS			VIN	1	2	87		6	AIN 3.3V		9	PRU1				
USB1		V_EN	GPIO	109	3	4	89				11	PRU				
USB1			VBUS	5	6	5	GPIO	CS	SPI0	TX	PRU					
			VIN	7	8	2		CLK		RX	UART2					
			DN	9	10	3		MISO		TX						
			DP	11	12	4		MOSI		RX	PRU					
			ID	13	14	3.3V		SYS								
			GND	15	16	GND										
AIN 1.8V			REF-	17	18	REF+	AIN 1.8V									
			0	19	20	20	GPIO		16(in)	PRU0						
			1	21	22	GND	SYS									
			2	23	24	VOUT	SYS									
			3	25	26	12		SDA	I2C2	TX	CAN0					
			4	27	28	13		SCL		RX						
PRU0	7	QEP0	STRB	GPIO	117	29	30	43	GPIO	TX	UART0	15	PRU1			
	4		A		114	31	32	42		RX		14				
	1	PWM0	B			111	33	34	26							
PRU1	10				88	35	36	110		A	PWM0	0	PRU0			

P2															
PWM1				A		50	1	2	59						
PWM2				B		23	3	4	58						
UART4				RX	GPIO	30	5	6	57	GPIO					
				TX		31	7	8	60						
CAN1		RX	I2C1	SCL		15	9	10	52						
		TX		SDA		14	11	12	PWR BTN	SYS					
SYS					VOUT	13	14	VIN	BAT						
					GND	15	16	TEMP							
GPIO						65	17	18	47	GPIO	STRB	QEP2	15i	PRU0	
						27	19	20	64						
SYS					GND	21	22	46	GPIO	IDX A	QEP2	14(in)	PRU0		
					3.3V	23	24	44				14(out)			
CAN1	RX	SPI1	MOSI		41	25	26	NRST	SYS						
	TX		MISO		40	27	28	116	GPIO	IDX	QEP0	6	PRU0		
PRU	eCAP		CLK		7	29	30	113				3			
PRU1	16(in)		CS		19	31	32	112	2						
PRU0	15(out)	QEP2	B		45	33	34	115		B	QEP0	5			
PRU1	8	AIN 3.3V	5		86	35	36	7	AIN 1.8V						

PocketBeagle Schematics



Simplified Layout



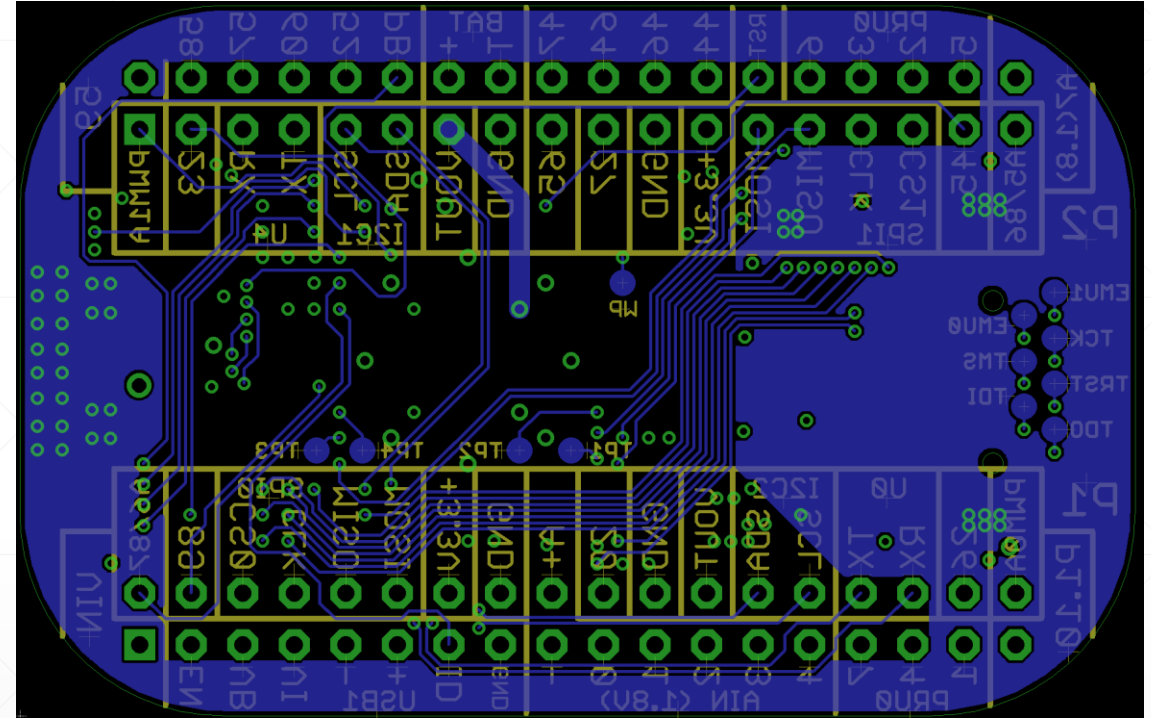
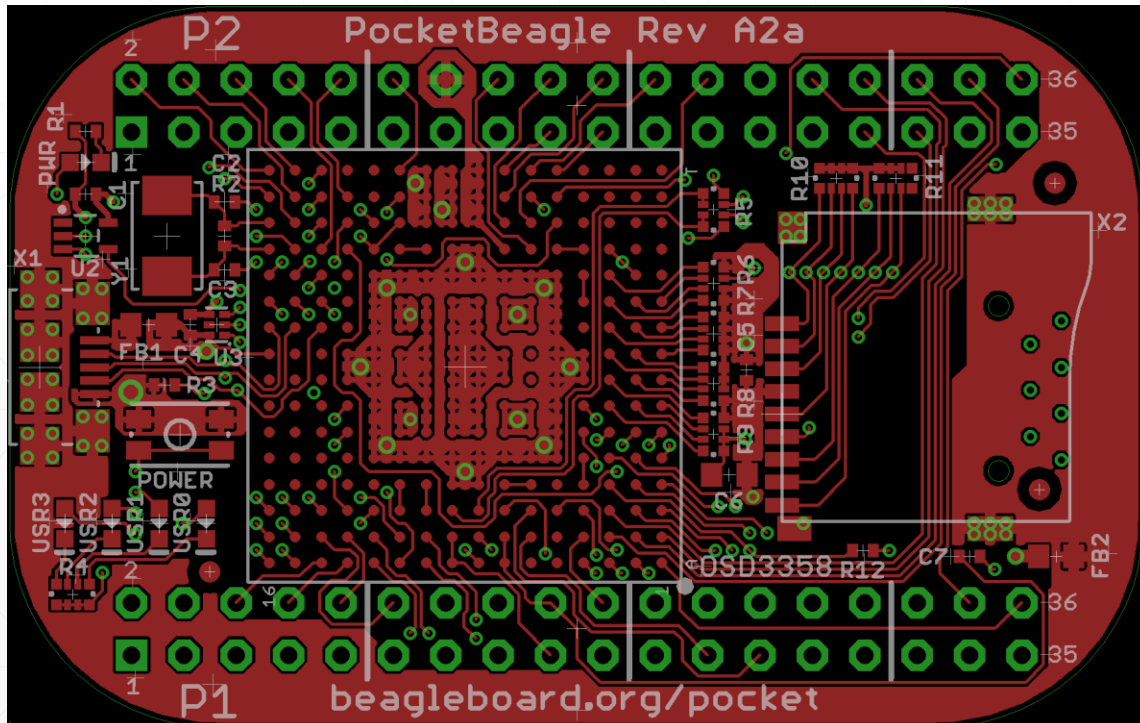
All Signals Escaped in a Single Layer

- ▶ 6 mil Trace Width
- ▶ 6 mil Space

All Power Domains and Internal Signals located in the center for easy connection

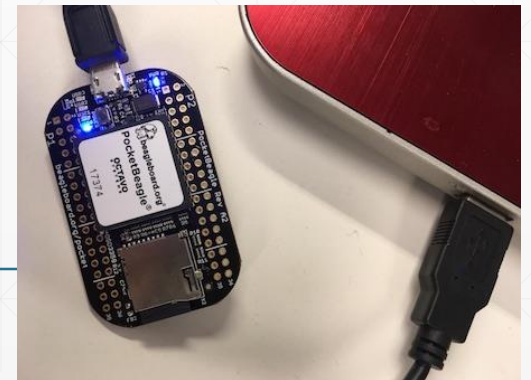
PocketBeagle Layout

- Open Source Schematics & Layout
- 4 layers PCB
 - 6 mil trace / 6 mil space
 - 15 mil drill / 25 mil via



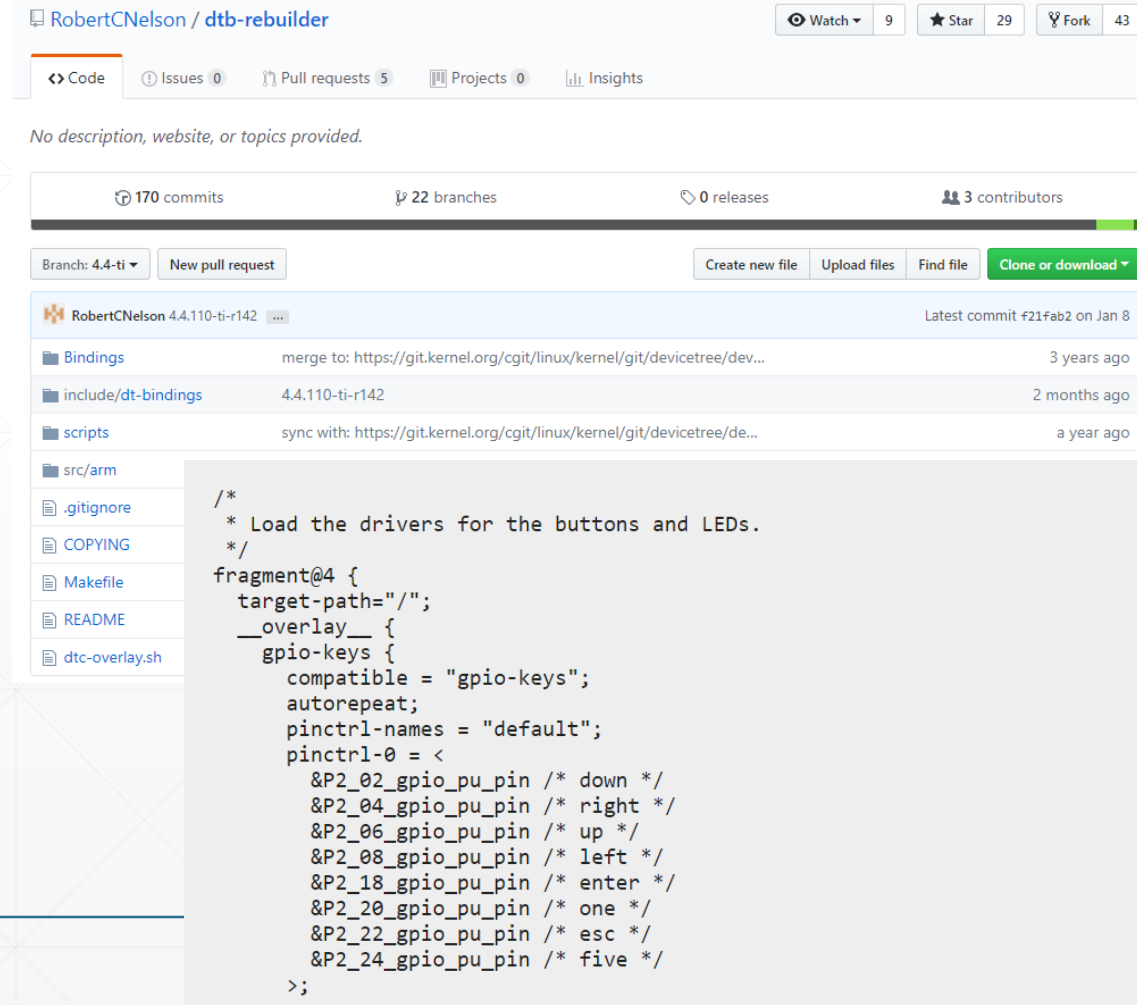
Simplified Board Bring Up Process

- Hardware Bring Up
 - Verify Power Isolation (ie your power rails are not shorted to ground – Don't release the magic smoke)
- Software Bring Up
 - Download Latest Image from BeagleBoard.org
 - Modify the device tree to meet your needs
 - Power up the board and check that everything boots properly
- You don't worry about
 - Bad voltages to the processor or DDR
 - DDR not working



Modifying Your Device Tree

- Development boards provide good device tree infrastructure
 - Majority of your device tree is already done for you
 - Only update the items that are different for your board
 - Many examples to mine for information / help
 - <https://github.com/RobertCNelson/dtb-rebuilder>
- Prototyping can be done with device tree overlay
 - Allows testing on your SBC prototyping board



RobertCNelson / dtb-rebuilder

Watch 9 Star 29 Fork 43

Code Issues 0 Pull requests 5 Projects 0 Insights

No description, website, or topics provided.

170 commits 22 branches 0 releases 3 contributors

Branch: 4.4-ti New pull request Create new file Upload files Find file Clone or download

RobertCNelson 4.4.110-ti-r142 Latest commit f21fab2 on Jan 8

Bindings	merge to: https://git.kernel.org/cgit/linux/kernel/git/devicetree/dev...	3 years ago
include/dt-bindings	4.4.110-ti-r142	2 months ago
scripts	sync with: https://git.kernel.org/cgit/linux/kernel/git/devicetree/de...	a year ago

src/arm

.gitignore

COPYING

Makefile

README

dtc-overlay.sh

```
/*
 * Load the drivers for the buttons and LEDs.
 */
fragment@4 {
    target-path="/";
    __overlay__ {
        gpio-keys {
            compatible = "gpio-keys";
            autorepeat;
            pinctrl-names = "default";
            pinctrl-0 = <
                &P2_02_gpio_pu_pin /* down */
                &P2_04_gpio_pu_pin /* right */
                &P2_06_gpio_pu_pin /* up */
                &P2_08_gpio_pu_pin /* left */
                &P2_18_gpio_pu_pin /* enter */
                &P2_20_gpio_pu_pin /* one */
                &P2_22_gpio_pu_pin /* esc */
                &P2_24_gpio_pu_pin /* five */
            >;
        };
    };
}
```

Using a SiP in your Linux Computer Design Will:

- Bring you 100+ components in one package
 - Makes board design faster, simpler and easier to add your own new features
 - Ensures easy board bring-up
 - Give you the heart of the Computer Hardware in a single BGA package
 - Lower cost PCB, fewer board layers, single sided
 - Easy to manufacture with - Some have even hand soldered it!
 - Bridge the gap between Prototype and Production
 - Open Hardware + Open Source Software
 - Easy migration from SBC prototyping board to your custom PCB
-



Thank You

For more information come to our table at ELC Technical Showcase